

# PATHCONF

Vulnerable to TOCTOU issues

Sean Barnum, Cigital, Inc. [vita<sup>1</sup>]

Copyright © 2007 Cigital, Inc.

2007-04-02

## Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 6894 bytes

<b>Attack Category</b>	<ul style="list-style-type: none"><li>• Path spoofing or confusion problem</li></ul>	
<b>Vulnerability Category</b>	<ul style="list-style-type: none"><li>• Indeterminate File/Path</li><li>• TOCTOU - Time of Check, Time of Use</li><li>• Unconditional</li></ul>	
<b>Software Context</b>	<ul style="list-style-type: none"><li>• File Path Management</li><li>• File Management</li></ul>	
<b>Location</b>		
<b>Description</b>	<p>The pathconf function is used to provide methods for the application to determine the current value of a configurable limit or option that is associated with a file or directory. The first input is the name of a file or directory and the second input is a constant that represents the configurable system limit or option to be returned.</p> <p>pathconf() is vulnerable to TOCTOU attacks. The existence of a call to this function should unilaterally be flagged.</p>	
<b>APIs</b>	<b>Function Name</b>	<b>Comments</b>
	pathconf	use
	lpathconf	use
<b>Method of Attack</b>	<p>The key issue with respect to TOCTOU vulnerabilities is that programs make assumptions about atomicity of actions. It is assumed that checking the state or identity of a targeted resource followed by an action on that resource is all one action. In reality, there is a period of time between the check and the use that allows either an attacker to intentionally or another interleaved process or thread to unintentionally change the state of the targeted resource and yield unexpected and undesired results.</p> <p>The pathconf() call is a use-category call, which when preceded by a check-category call can be indicative of a TOCTOU vulnerability.</p>	

1. <http://buildsecurityin.us-cert.gov/bsi-rules/35-BSI.html> (Barnum, Sean)

Exception Criteria				
Solutions		Solution Applicability	Solution Description	Solution Efficacy
		Applicable to all occurrences.	As with all occurrences of TOCTOU-vulnerable function API pairs (one char* filename version and one file descriptor version) given equivalent (or better with the fd version), all occurrences of the filename version (in this case pathconf) of the function should be replaced with the fd version, fpathconf.  If there is some reason that this replacement strategy can't occur, standard TOCTOU avoidance techniques should be performed.	Effective
		Generally applicable.	The most basic advice for TOCTOU vulnerabilities is to not perform a check before the use. This does not resolve the underlying issue of the execution of a function on	Does not resolve the underlying vulnerability but limits the false sense of security given by the check.

		a resource whose state and identity cannot be assured, but it does help to limit the false sense of security given by the check.	
	Generally applicable.	Limit interleaving of process access to the filename.	Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.
	Generally applicable.	Limit the time/code distance between the check and the use	Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.
	Generally applicable.	Recheck the resource after the use call to verify that the action was taken appropriately.	Effective in some cases.
<b>Signature Details</b>		long fpathconf(int fildes, int name); long pathconf(const char *path, int name);	
<b>Examples of Incorrect Code</b>		<pre>char filename[]="safefile.txt"; int theName=safeValue; strcpy(filename,"unsafefile.txt"); theName=unsafeValue; pathconf(filename,theName);</pre> <p>In this case, the operation performed by the pathconf function is not what was intended.</p> <p>Since the parameters of a file are being reset, it is also possible for a race condition to be created.</p>	
<b>Examples of Corrected Code</b>			
<b>Source References</b>		<ul style="list-style-type: none"> <li>Viega, John &amp; McGraw, Gary. <i>Building Secure Software: How to Avoid Security Problems the Right Way</i>. Boston, MA: Addison-Wesley Professional, 2001, ch. 9.</li> <li>UNIX man page for pathconf()</li> </ul>	

	<ul style="list-style-type: none"> <li>• UNIX man page for fpathconf()</li> <li>• Bishop, Matt &amp; Dilger, Michael. “Checking for Race Conditions in File Accesses.” <i>Computing Systems</i> 9, 2 (Spring 1996): 131-152.</li> </ul>	
<b>Recommended Resource</b>		
<b>Discriminant Set</b>	<b>Operating System</b>	<ul style="list-style-type: none"> <li>• UNIX</li> </ul>
	<b>Languages</b>	<ul style="list-style-type: none"> <li>• C</li> <li>• C++</li> </ul>

## Cigital, Inc. Copyright

Copyright © Cigital, Inc. 2005-2007. Cigital retains copyrights to this material.

Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.

For information regarding external or commercial use of copyrighted materials owned by Cigital, including information about “Fair Use,” contact Cigital at [copyright@cigital.com](mailto:copyright@cigital.com)<sup>1</sup>.

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. <mailto:copyright@cigital.com>